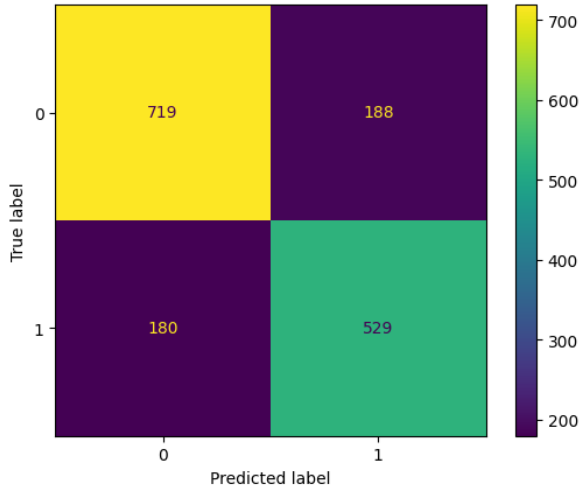
We can see from the confusion matrix above that the model is balanced, there is not a category that the model constantly made errors.



Then if we use XOR to map out the predicted value from the testing value we spitted, we noticed there are 358 errors.

```
from collections import Counter
Counter(xor_mask)
```

[95] ✓ 0.3s

```
Counter({False: 1258, True: 358})
```

I tried to print out all of the columns which the model predicted falsely.

	body_score	inlinks_scaled	answer_type	paren_match	obs_len	tournaments	aws	predicted_val
4992	120.714627	0.389657	6	0	561	16	0	1
433	57.972990	0.000000	8	0	556	41	1	0
2079	33.870023	0.167875	0	0	177	37	0	1
5061	76.154803	0.194828	8	0	657	17	1	0
4934	49.128107	0.167875	6	0	516	49	0	1
...	...	...	...	...	...	...	...	...
4525	46.042435	0.205864	0	0	288	64	0	1
33	29.807640	0.327597	8	1	235	4	1	0
7457	113.306061	0.280678	6	0	684	34	1	0
4489	42.441320	0.459588	6	0	361	64	0	1
4933	51.875005	0.129886	7	0	516	49	0	1

358 rows × 8 columns

Then I tried to group the data frame by the answer_type.

answer_type	count
6	169
8	112
0	63
3	6
7	3
1	2
5	2
2	1

As we can see the model did not do well on answer types 6,8,0, and 3.

If we combine the count with the original testing data, we can see that the on ‘anim’, the error percentage is the highest with an error rate of 33 percent.
But considering the count of ‘anim’ is few, it will make more sense if we try to improve the category of ‘people’ with the question amount of 607.

answer_type	count	answer_type_by_catagory	all_count	error_percentage
2	1	anim	3	0.333333
6	169	people	607	0.278418
8	112	work	570	0.196491
0	63	None	322	0.195652
1	2	abs	12	0.166667
7	3	place	19	0.157895
5	2	org	18	0.111111
3	6	char	58	0.103448